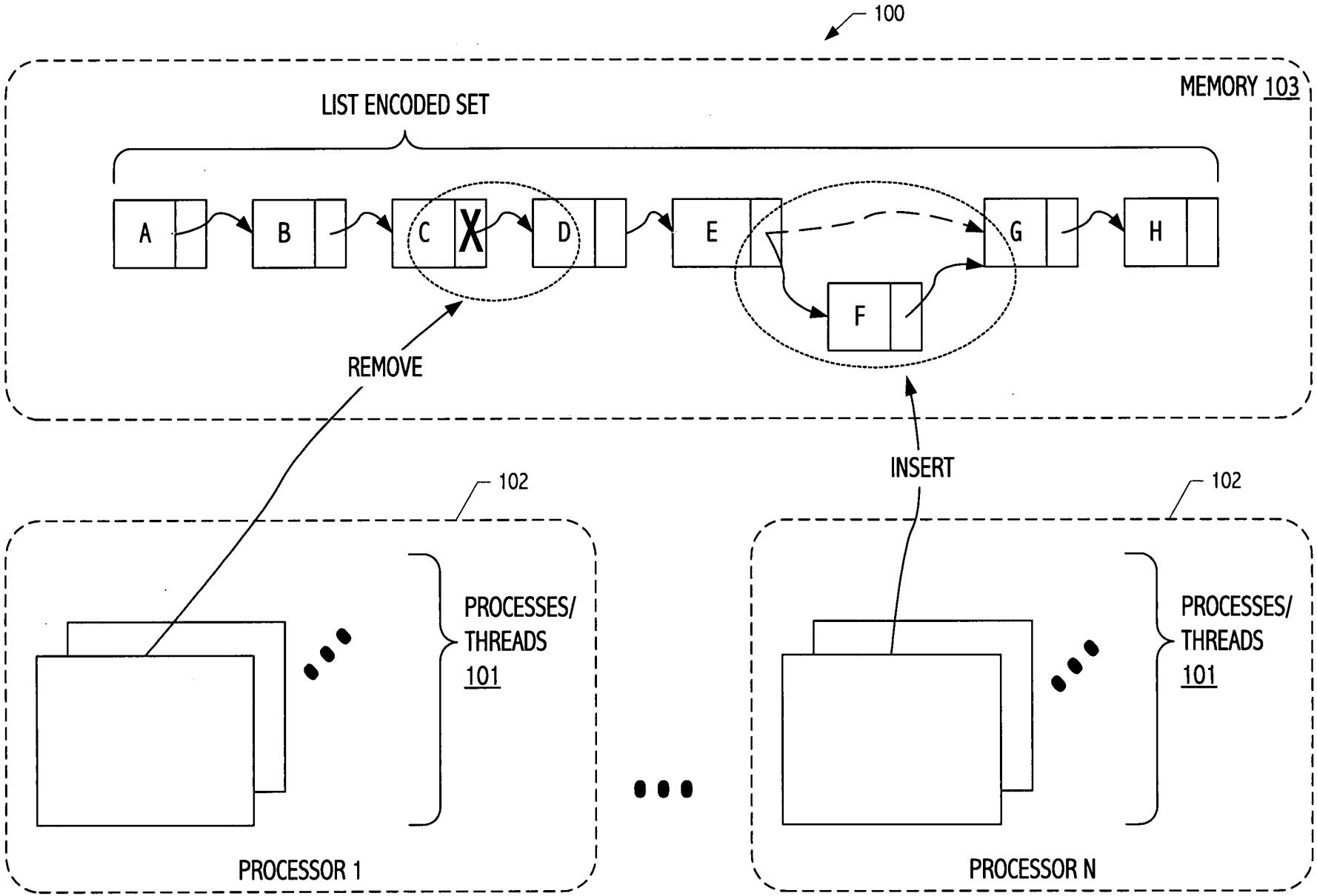
100
MEMORY 103
LIST ENCODED SET
A
B
C
X
D
E
F
G
H
REMOVE
INSERT
102
102
PROCESSES/
THREADS
101
PROCESSES/
THREADS
101
PROCESSOR 1
PROCESSOR N

FIG. 1

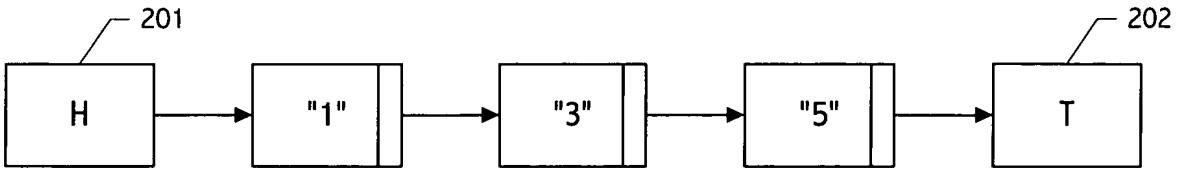

FIG. 2

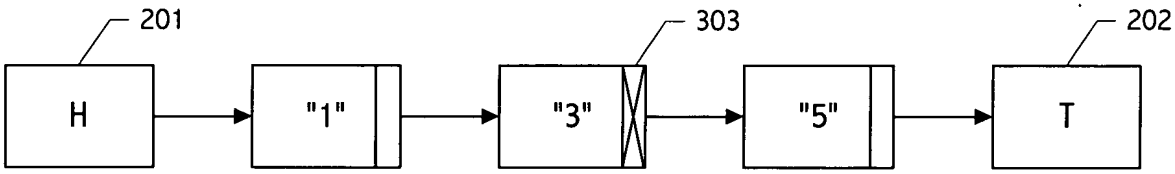

FIG. 3A

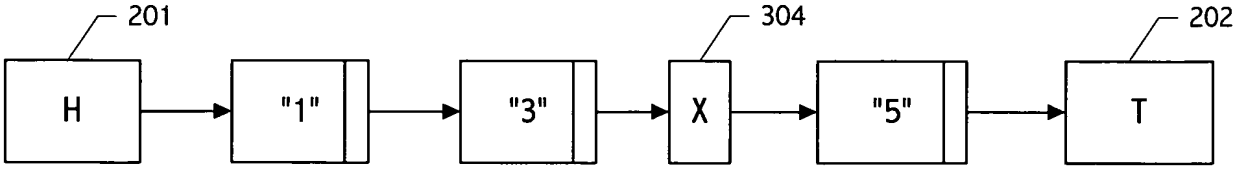

FIG. 3B

```
bool List::insert(KeyType key) {
  Node *new_node, *right_node, *left_node;

  new_node = NULL;
  do {
    right_node = search (key, &left_node);

    if ((right_node != tail) && (right_node->key == key)) {
      return false;
    } else {
      if (new_node == NULL) new_node = new Node(key);
      new_node->next = right_node;
      if (CAS (&(left_node->next), right_node, new_node)) /*C2*/
        return true;
    }
  } while (true); /*B3*/
}
```

C2

B3

FIG. 4

```
bool List::remove(KeyType search_key)

  Node *right_node, *right_node_next, *left_node;

  do {
    right_node = search (search_key, &left_node);

    if ((right_node == tail) | |
        (right_node->key != search_key)) return false;
    right_node_next = right_node->next;
501 if (!is_marked_reference(right_node_next))
      if (CAS(&right_node->next),     /* C3 */
            right_node_next,                                  C3
              get_marked_reference(right_node_next)) {                    C4
          if (!CAS(&(left_node->next), right_node, right_node_next))  /* C4 */
502         search(right_node->key,&left_node);
          return true;                                              503
        }
    } while (true); /*B4*/
  }                                  B4
```

FIG. 5

```
bool List::find (KeyType search_key)

  Node *right_node;
  Node *left_node;

  right_node = search (search_key, &left_node);
  return ((right_node != tail) && (right_node->key == search_key)) ;
}
```

FIG. 6

```
Node* List::search (KeyType search_key, Node **left_node) {
 Node *left_node_next;
 Node *right_node;

search_again:
 do {
  Node* t = head;
  Node* t_next = head->next;

  /* 1: Find left_node and right_node */

  do {
   if (!is_marked_reference(t_next)) {
    (*left_node) = t;
    left_node_next = t_next;
   }
   t = t_next;
   if (t == tail) break;
   t_next = t_next->next;
  } while (is_marked_reference(t_next) || (t->key < search_key)); /*B1*/
  right_node = t;

  /* 2: Check if right_node is the immediate successor of left_node */

  if (left_node_next == right_node)
   if ((right_node != tail) && is_marked_reference(right_node->next))
    goto search_again; /*G1*/
   else
    return right_node; /*R1*/

  /* 3: Remove one or more marked nodes between left_node and right_node */

  if (CAS (&((*left_node)->next), left_node_next, right_node)) /*C1*/
   if ((right_node != tail) && is_marked_reference(right_node->next))
    goto search_again; /*G2*/
   else
    return right_node; /*R2*/
 } while (true); /*B2*/
}
```

701 702 703 B1 G1 R1 C1 G2 R2 B2

FIG. 7

```
Node* List::deleteGE (KeyType search_key)

 Node *right_node, *right_node_next, *left_node;

 do {
  right_node = search (search_key, &left_node);

  if (right_node == tail) return false; /*T1*/          T1

  right_node_next = right_node->next;
  if (!is_marked_reference (right_node_next)) {
   if (CAS (&(right_node->next),
         right_node_next,
         get_marked_reference (right_node_next))) { /*C5*/      C5
      if (!CAS (&(left_node->next), right_node, right_node_next)) /*C6*/      C6
       search (right_node->key, &left_node);

      return right_node;
   }
  }
 } while (true); /*B4*/          B4
}
```

FIG. 8

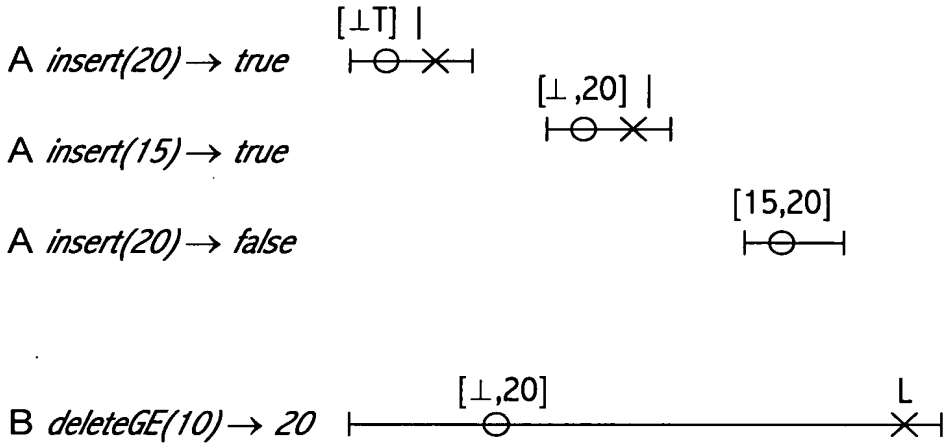
[⊥T] |
A insert(20) → true
[⊥,20] |
A insert(15) → true
[15,20]
A insert(20) → false
[⊥,20]
L
B deleteGE(10) → 20

FIG. 9

```
bool List::insert2 (KeyType key) {
 Node new_node*, *dup_right_node, *right_node, *left_node;

 newNode = NULL;
 do {
  right_node = search (key, &left_node);

  if ((right_node != tail) && (right_node->key == key))
   return false;
  else {
   if (new_node == NULL) new_node = new Node(key);
   if (right_node == tail) {
    new_node->next = tail;
    if (CAS (&(left_node->next), tail, new_node)) /*C7*/
     return true;                                          C7
   } else {
    Node *right_node_next;
    dup_right_node = new Node (right_node->key);
    new_node->next = dup_right_node;
    right_node_next = right_node->next;
    if (!is_marked_reference (right_node_next)) {
     dup_right_node->next = right_node_next;
     if (CAS (&(right_node->next),
           right_node_next,
           get_marked_reference (dup_right_node))) { /*C8*/
       if (!CAS (&(left_node->next),                          C8
            right_node,
            new_node)) /*C9*/
         search (key, &left_node);                            C9
       return true;
     }
    }
   }
  }
 } while (true); /*B5*/
}                                                             B5
```

FIG. 10

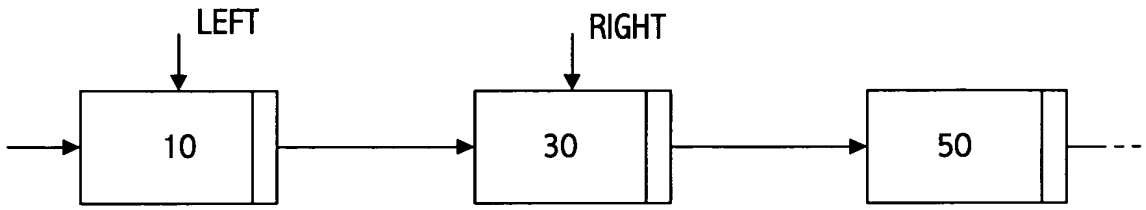

FIG. 11A

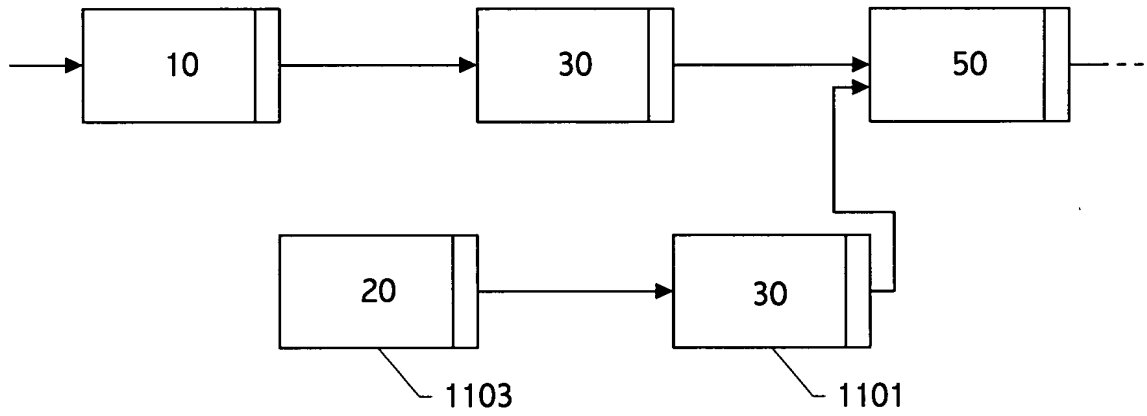

FIG. 11B

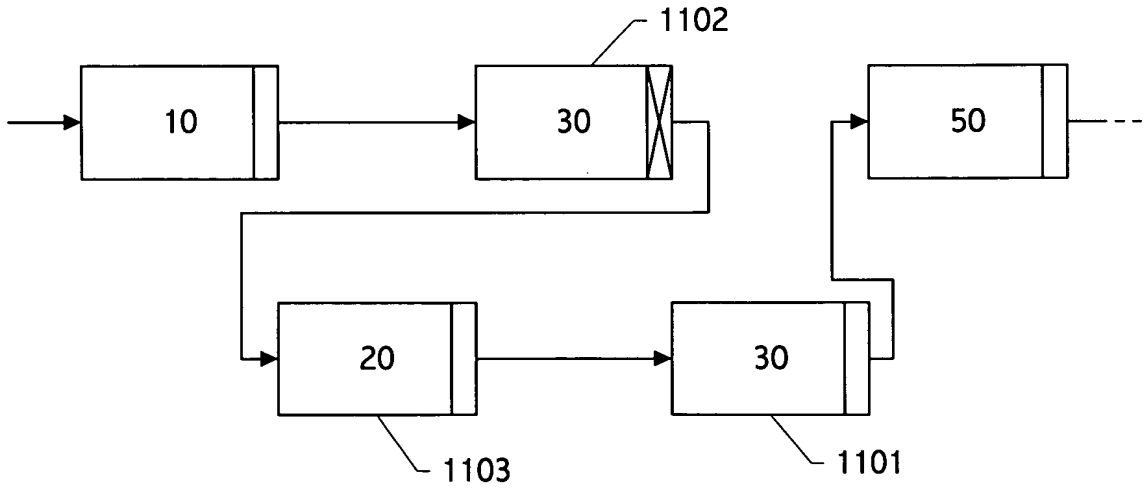

FIG. 11C